Proceedings of the 2010 14th International Conference on Computer Supported Cooperative Work in Design

HLMP API: A Software Library to Support the Development of Mobile Collaborative Applications

Juan Rodríguez-Covili, Sergio F. Ochoa, José A. Pino
Department of Computer Science
Universidad de Chile
Santiago, Chile
{jrodrigu, sochoa, jpino}@dcc.uchile.cl

Roc Messeguer, Esunly Medina, Dolors Royo
Department of Computer Architecture
Universitat Politècnica de Catalunya
Barcelona, Spain
{messeguer, esunlyma, dolors}@ac.upc.edu

***Abstract*—Mobile collaborative applications are usually deployed in work scenarios where the existence of fixed communication infrastructure is hard to predict. For that reason, these applications use Mobile Ad hoc Networks (MANETs) to support communication between mobile users. The complexity involved in such communication infrastructures make that developers avoid developing software for mobile work scenarios. However, it is possible to provide a reusable abstraction of such communication mechanisms, in order to avoid that developers have to deal with low-level programming. This article presents HLMP API, which is an application programming interface that provides access to a HLMP implementation. This API is organized as a fully distributed mobile communication infrastructure, able to run on MANETs. This infrastructure provides an important set of services, which are required to support mobile collaboration. The reuse of these services allows developers to reduce the complexity, times and cost of these development projects.**

Keywords-MANET; mobile collaborative work; mobile collaborative applications; communication support; collaborative systems development.

I. INTRODUCTION

The concept of Mobile Ad hoc Network (MANET) has become an interesting contribution to solve communication problems on loosely coupled activities, carried out in several mobile collaborative work scenarios [4, 17]. In those contexts, the applications are not usually reliant on a fixed infrastructure communication system, such as antennas or access points. A MANET creates a communication mesh to exchange messages among participating devices. Those devices are free to move, change connection status and even support autonomous tasks. However, the successful development of mobile collaborative applications usually has to mainly deal with network related issues, instead of focusing efforts on social groupware architectures and functionalities [5, 14]. This occurs because collaboration and coordination problems are highly dependent on the computer-mediated communication [6, 14, 18]. Therefore, it is important to count on an abstract solution, which allows developers to reuse the networking services without having to deal with low-level programming details. This reuse can be done through an application programming interface (API) providing access to particular implementation of several communication and coordination services.

The authors have previously described the design and implementation of a routing protocol called High Level MANET Protocol (HLMP) [22]. Such proposal implements a set of services required by mobile collaborative applications when they are supported by a MANET. Examples of these services are MANET formation, IP addresses self-configuration and duplication detection, peers discovery, and routing for multicast and unicast messages. Moreover, the authors have designed a number of mobile shared workspace applications for various scenarios, namely, construction inspection activities, hospital healthcare work and urban emergency response [15, 17, 19, 21].

This implementation process experience has also shown that mobile groupware development requires reusable interfaces for the communication mechanisms. The goal is to save cost, time and effort on network related problems, concentrating instead on the development of supporting collaborative activities. The final objective is thus to improve productivity and software quality [20, 23]. An application programming interface seems to be a good idea to encapsulate the networking services, and provide developers with an abstract access to those functionalities. Such API could provide useful information to collaborative applications, allowing the implementation of awareness mechanisms. Those mechanisms would provide detection of, e.g., users' connection and availability, or mobile workers' location [14].

This paper presents HLMP API, an implementation of HLMP which enhances mobile groupware development with a high level API. This interface keeps the abstraction of the communication processes through a message exchange paradigm and an event delivery method. This reusable infrastructure supports communication functionalities and it takes advantage of the HLMP specification. In order to show the quality of the solution that is being reused, this article presents a performance comparison between HLMP API and an implementation of the OLSR protocol [3]. This last protocol is one of the well-known and currently used routing protocols for MANETs. The tests were carried out using a peer to peer file transfer routine in static and mobile scenarios.

Next section describes related work. Section III presents the HLMP API structure and its components. Section IV shows the performed tests, the obtained results and it also provides

978-1-4244-6763-1/10/$26.00 ©2010 IEEE

comparison between the studied routing implementations. Finally, section V presents the conclusions and future work.

II. Related Work

Several initiatives propose reusable functions to support collaboration in peer-to-peer networks. One of them is LaCOLLA [12]. This middleware has a fully decentralized peer-to-peer architecture and provides general purpose functionalities for building collaborative applications. However, this middleware requires networks with important signal stability, it does not provide routing and it is not able to run on hardware with scarce resources.

A similar framework is iClouds, which offers spontaneous mobile user interaction and file exchange support in MANETs [9]. Unfortunately this platform is focused just on file sharing.

Other frameworks providing specific functionalities to support mobile collaboration through an API are YCab [2] and JXTA [11]. Although these platforms have shown to be useful to support collaboration in peer-to-peer networks, they require signal stability. Therefore, they are unsuitable to be used on ad-hoc mobile work settings.

Nokia is an interesting protagonist. Such company developed a services-oriented framework that could be used to support mobile collaboration. This framework includes a set of APIs and an SDK (Software Development Kit) allowing developers to create service-oriented applications that act as consumers of Web services on mobile devices [10]. Since mobile applications can just consume services, their autonomy is small because they require a service provider, which is unsuitable for MANETs.

Finally, there are several proposals to share information in P2P networks, even considering mobile computing devices [8, 17]. Typical examples of these platforms are the tuple-based distributed systems derived from LINDA, such as: FT-LINDA, JINI, PLinda, T-spaces, Lime, JavaSpaces and GRACE [1, 7, 16]. All these solutions use centralized components; therefore they cannot be used on MANETs. XMIDDLE [13] is another middleware allowing mobile hosts to share XML documents across heterogeneous mobile hosts, permitting on-line and off-line access to data. Nevertheless, these middleware are just focused on data sharing and they do not support the autonomy and interoperability capabilities required by mobile workers.

III. HLMP API

HLMP API is composed of two main components (Fig. 1): a *core* and *plug-ins*. On the one hand, the core implements the mechanisms to support the communication process, network data interchange and operating systems interoperability procedures for delegated functionalities. On the other hand, the plug-ins contain specifications for structuring groupware communication protocols and awareness mechanisms, which use the services provided by the core. Next subsections explain these two components in detail.

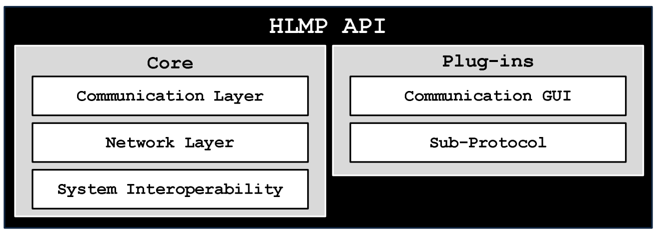

Figure 1. HLMP API structure

A. HLMP Core

The core is the basic structure of the HLMP implementation. It performs the configuration of threshold values, it establishes the MANET connection procedures, and structures, routes and it delivers messages through the network. This component also keeps control on the events triggered by the collaborative applications connected to the MANET. Such information is reported to the application upper layers, which are in charge of implementing the awareness mechanisms which help mobile users to collaborate on-demand.

The core is divided into three functional sub-structures: *system interoperability*, which is the procedure bus for operating systems (OS) delegated functions; *network layer*, which is responsible for the exchange of UDP and TCP services and datagrams; and *communication layer*, which is the API potentially used by developers as support for mobile collaborative applications.

1) System Interoperabilty

This layer is in charge of specific OS delegated functions. These functions include the actions required to carry out the MANET connection procedure, i.e. configuration of the WLAN profile in an XML form, management of the wireless network adapter, configuration of the IP address and subnet mask, and detection of duplicated address when connecting to the specified WLAN, through OS notifications.

2) Network Layer

This component implements the TCP and UDP services required to exchange messages in a MANET. These messages are validated and queued, while receiving, to be lately attended by the Communication Layer. It is also the component that manages the links with the remote devices neighborhood using TCP direct connections.

3) Communication Layer

This module implements the application programming interface that developers can use to support communication in collaborative mobile applications. This component is also a manager of the HLMP services, i.e. routing mechanisms, messages organization, message packing and unpacking. The main concept relays on attending the queue of network messages received through the Network Layer, and using a factory to transform them into communication messages. Then, the system decides what action to take once it has identified the messages. Typically these actions are routing or attending them. There is also a queue for outcoming messages which are sent by the groupware layers. This queue is processed in order to transform messages into the corresponding byte packets that are finally sent through the network.

Fig. 2 shows how the communication messages are organized. The API uses the requirements of HLMP for creating a composed object-oriented specification. The main abstract class *Message* contains the basic information of a message, i.e. identification number, meta-type code, message type code, sub-protocol type code, information about the sender and the number of hosts through which the message has been routed. From this class, three main abstract classes are derived. These classes represent the three Meta Types of messages defined in HLMP, i.e., Multicast Message, Unicast Message and Safe Unicast Message.

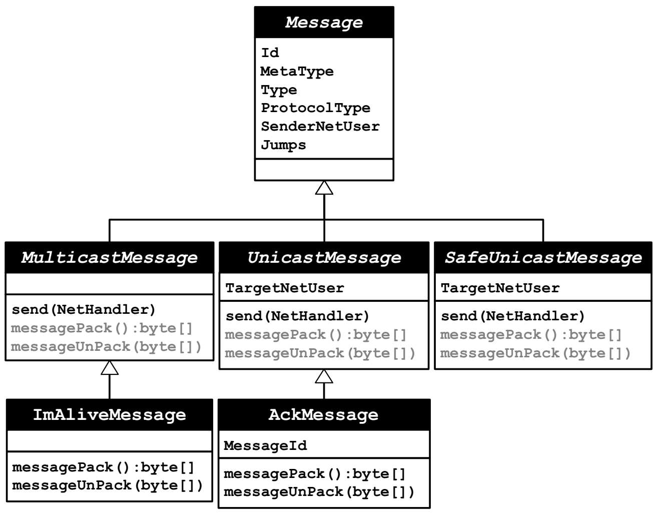

Figure 2. Message class context diagram

Every Meta Type message decides how to send a packet, by using the appropriate functionality offered by the Network Layer. This delivery also considers the semantic mechanism specified for each type of delivery. For example, multicast messages are sent using the UDP channels to every user in the MANET. Unicast messages are sent only to one remote user, identified by the target user property, but using the TCP channels. Finally, Safe Unicast messages are sent only to one user with the same mechanism that the previous ones. However, the sender waits for a confirmation of the reception message (i.e., an ACK) in this case. If the ACK is not received within a certain time period, the message is sent again.

The developers must extend and implement classes derived from the Meta Type messages in order to build new kind of messages for specific groupware requirements. Examples of these extensions are the internal construction of the "I'm Alive" message, used to implement peers' detection mechanisms, and the "Ack" message, used for notifications of received Safe Unicast messages.

It is also important that every implemented message is in charge of the procedures for reading and writing its own byte message packets, depending on the information it needs to propagate. The Meta Types use polymorphism to call those procedures when creating the message bytes packets.

Fig. 3 shows the main components of the communication process, i.e. the Communication class. This class offers the functionality for connecting or disconnecting to/from the MANET, and also to send any kind of derived message.

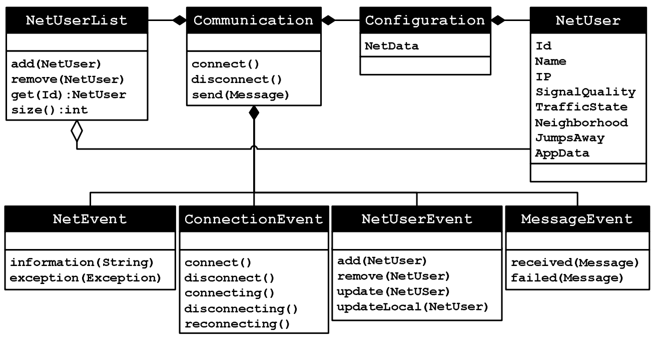

Figure 3. Communication class context diagram

A user in the MANET is managed as a NetUser. This class has all the properties required to identify and keep the information about the user signal quality, traffic state, semantic information like the name, identification number or other data required by groupware applications, e.g., group association, position relative to a coordinates system, and system permissions. The local user information is kept in the Configuration object along with local network parameters. The rest of the MANET users are kept in a NetUserList object, which is managed internally by the Communication class.

MANET events are communicated to collaborative applications, because these systems typically implement awareness based on such information. NetEvent emits notifications about the internal behavior of the system, such as log information and exceptions. ConnectionEvent manages the notifications triggered when the connection status of the local user has changed or it is currently changing. NetUserEvent triggers events related to the rest of the users in the MANET. For example, events notifying that new users are connected to the network or that a mobile user goes to an offline status. This information is also updated in the NetUserList object.

Finally, the MessageEvent component manages the messages that were received and accepted; and that need to be processed as received messages. It also manages the Safe Unicast messages which were sent by the groupware layer and not delivered successfully to destination, due to disconnection of the target user or failure when trying to find a path to destination.

B. HLMP Plug-ins

The API offers, as an optional feature, an organization for sub-protocols implementations (e.g. groupware specific protocols), involving more complex services, e.g. text messaging, mail boxes, files sharing and transfer, or data synchronization mechanisms.

These complements also offer fundamental mobile groupware communication GUIs. Fig. 4 shows the *SubProtocolI* interface, which can be instantiated in order to create a new sub-protocol that uses the HLMP Core. These objects are added to a SubProtocolList, managed by the Communication object. Whenever messages of a specific sub-protocol type are received, then the Communication class assigns that message to the corresponding instantiated sub-protocol. HMLP API presents some pre-built basic sub-protocols, such as a text message delivery protocol, for group and private communications; and a file transfer protocol, which allows network users to share and download any type of files

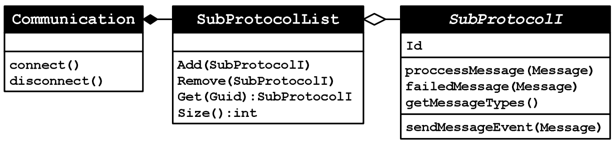

Figure 4. Sub-Protocol context diagram

Pre-built communication user interfaces for mobile collaborative applications are also accessible, i.e. icons representing users' list or connection awareness mechanisms. Fig. 5 (a) shows a user list which makes use of the Communication object to manage visual users' awareness. The first user shown in black is the local one. The color intensity of the participant's icon indicates the communication quality with that user: the darker the better.

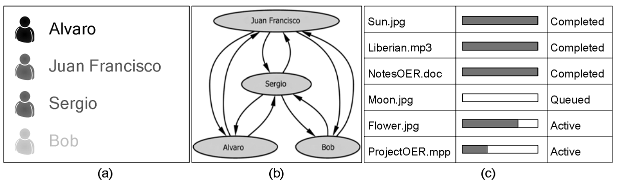

Figure 5. Common communication interfaces

Fig. 5 (b) shows a MANET graph representing the current networking links among mobile users. It is useful for showing the paths that messages can use for reaching their destination. It can also be used to see mobile network evolution and users' behavior.

Finally, Fig. 5 (c) shows the GUI corresponding to the file transfer sub-protocol. It illustrates the list of transferred files and their current status.

IV. Performance Comparison

The experimentation process involved three settings. In each setting, the HLMP API implementation was compared with an implementation of the OLSR protocol [3]. The reason to choose such protocol as a comparison instrument was because OLSR have one of the best routing implementations available for MANETs. The test scenarios used in the experimentations involved stationary and mobile nodes in a controlled scenario, as a way to reproduce the communication conditions of each setting. Next sections describe the evaluation scenarios and the obtained results.

A. *Test Case I*

In this case all nodes were located to one hop of distance. Initially there were just two nodes in the MANET, then a third one was added, then a fourth one, and so on to complete seven nodes (Fig.6).

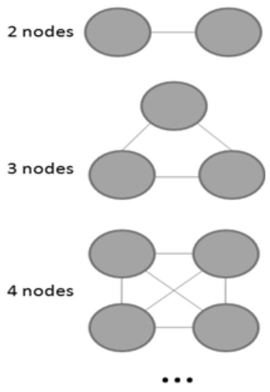

Figure 6. Evaluation Scenario I

In this case, the test measured the control overhead involved in each routing protocol (i.e. HLMP and OLSR). The obtained results (Fig. 7) show that HLMP packet sizes are similar to OLSR datagram sizes. OLSR packet seems to have a higher incremental proportion when new nodes are added. However, Fig. 8 shows that HLMP needs to send more packets when the nodes number increases in a group scenario. Despite, the great majority of those packets are discarded and dropped when performing the HLMP duplication detecting mechanisms.

Although the external behavior differences are not significant, such situation has a simple explanation: HLMP was designed to deal with high mobility of the users. For that reason the protocol need to monitor the network more frequently than OLSR, which produce the extra overhead at low levels, but that routine is controlled at higher levels.

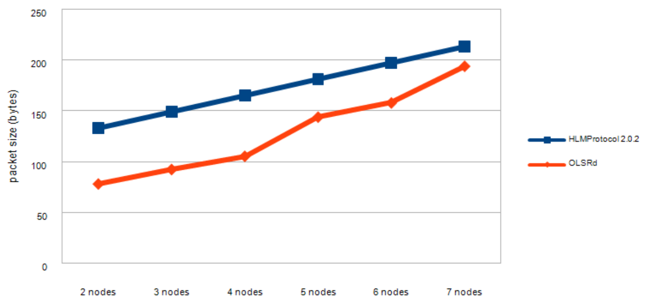

Figure 7. Test Case I: packet size

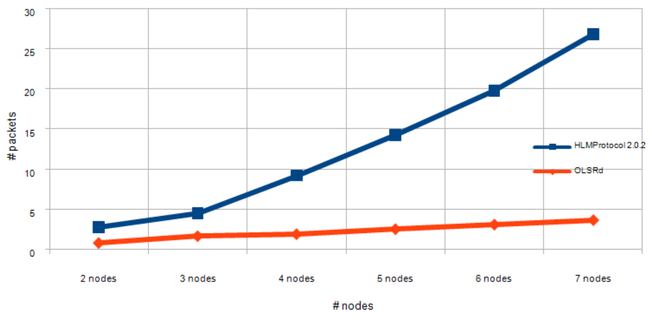

Figure 8. Test Case I: control overhead

B. Test Case II

The second test scenario involved the transmission of a file between a sender (Tx) and a receiver (Rx). Initially the MANET involved two nodes; then interim nodes were added in a line until complete three hops (Fig. 9). Two different files were used in this experiment. The first one weighted 993 KB and the second one weighted 2.7 MB.

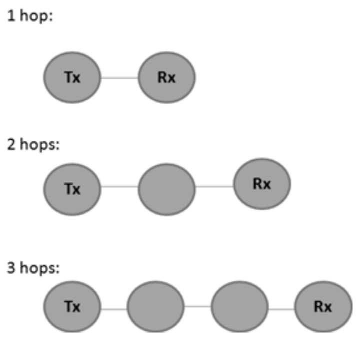

Figure 9. Evaluation Scenario II

Moreover, two OLSR implementations (for Windows and Linux respectively) were used in this test case, and one implementation of HLMP for Windows. The transfer overhead (bytes) related to each protocol implementation was measured, and the obtained results are presented in Fig. 10.

The results show that HLMP and OLSR have a similar transfer overhead, if we consider the same operating system. Considering different operating system, the HLMP overhead is 30% less than the OLSR overhead. This is because the transportation layer depends on the operative system protocol implementation for OLSR. HLMP is an independent platform.

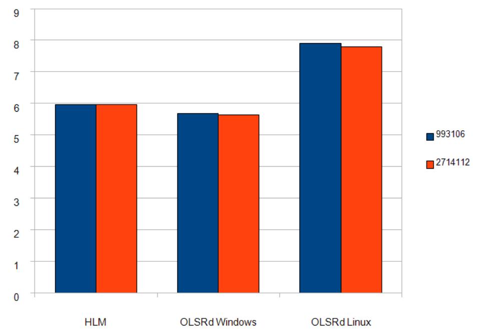

Figure 10. Test Case II: traffic overhead

C. Test Case III

Finally, the last scenario involves the transmission of a file between two remote nodes, but in a mobility context. The main nodes where situated statically in separated rooms. And five moving nodes where commutated between, creating active routes every 30 seconds (Fig. 11).

In this experimentation was measured the behavior of the data transmission and the impact of the MANET topology changes in each protocol. When a commutation of the interim nodes occurs, the network seems to have pauses to reactivate or recalculate the paths of the sent packets.

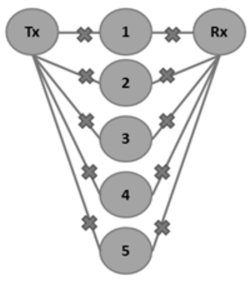

Figure 11. Evaluation Scenario III

Fig. 12 shows an average sample of a network pause in the OLSR implementation routine, when the system is reacting to connections and disconnections of the devices. Fig. 13 shows the same sample, but for the HLMP API. In all repeated sequences the OLSR protocol seems to have longer inactivity periods (13-15 seconds) than the HLMP system (lower than 10 seconds).

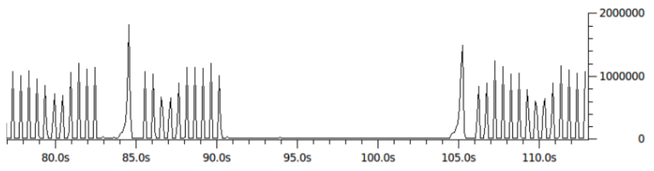

Figure 12. Test Case II: OLSRd raction behavior

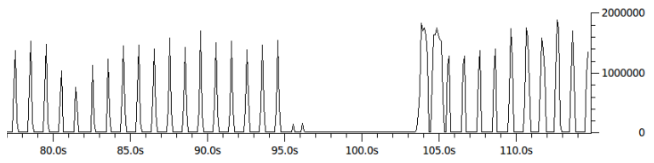

Figure 13. Test Case III: HLMP API reaction behavior

D. Discussion

The experimental results show that HLMP and the OLSR are similar in terms of routing performance and throughput. However, HLMP reacts faster to topology changes than OLSR due to the overhead cost to monitoring the network and the propagation of control packets ("I'm Alive" messages). Moreover, HLMP API implements the protocol in the application layer, which provides an ample control to decide when deliver the messages or the packets size to be used in each case.

Contrarily, the OLSR works inside the networking layer; therefore any mobile collaborative application that uses this protocol has to assume the parameters established by operating system when performing data transportation procedures. It reduces the capability to do tuning to collaborative solutions.

Considering the HLMP API performance and services, it is clear this is a component that worth of reusing. Developers of mobile collaborative applications can take advantage of this infrastructure.

V. Conclussions and Future Work

This paper has presented the HLMP API, a particular implementation of the HLMP routing protocol [22]. This communication infrastructure avoids that developers of mobile collaborative applications have to deal with low level communication details when develop a new software system. The reuse of the HLMP API services allows them to be focused on the groupware design aspects.

The upper layers of this API are able to create collaborative work functionalities by using the message exchange paradigm. It is also possible to reuse or implement network awareness mechanisms due to the several event types that are triggered to the collaborative application when important information or network behaviors occur.

The tests conducted to the API have shown the performance of this infrastructure is comparable to those obtained using OLSR protocol. If we consider that OLSR is one of the best protocols for MANETs, it is clear that the HLMP API worth of reusing. Typically this reuse helps to reduce the projects complexity and development times and cost. Moreover, it helps improve the quality of the final products.

Next steps for this work consider including new context-aware mechanisms to the API. Thus, the authors will intend to combine and adapt the API threshold values to increase the network performance or reduce excess of control overhead. These values can be grouped and configured depending on the work context. The API can inform to the collaborative application about changes in certain variables; therefore the application will implement self-adaption processes according to the work context changes.

ACKNOWLEDGEMENTS

This work was partially supported by Fondecyt (Chile), grants Nº 11060467 and 1080352, LACCIR grants No. R0308LAC004 and R0308LAC005, and Spanish MEC project P2PGrid TIN2007-68050-C03-01.

REFERENCES

[1] Bosneag, A.M., Brockmeyer, M.: "GRACE: Enabling collaborations in wide-area distributed systems"; Proc. of WETICE'05, Workshop on Distributed and Mobile Collaboration (DMC), IEEE CS Press, Linkoping University Sweden, 72-77, 2005.

[2] Buszko, D., Lee, W., Helal, A.: "Decentralized Ad-Hoc Groupware API and Framework for Mobile Collaboration"; Proc. of ACM Int. Conf. on Supporting Group Work (GROUP'01), ACM Press, Colorado USA, 5-14, 2001.

[3] T. Clausen, P. Jacquet: "Optimized Link State Routing Protocol (OLSR)". IETF RFC 3626, October 2003.

[4] Corson, S. Macker, J.: "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations". IETF, RFC 2501, January 1999.

[5] Dyck, J.: "A Survey of Application-Layer Networking Techniques for Real-time Distributed Groupware". Technical Report HCI-TR-06-06, University of Saskatchewan, 2006.

[6] Ellis, C.A., Gibbs, S.J., Rein, G.L.: "Groupware: Some Issues and Experiences". Communications of the ACM 34(1), 38–58, 1991.

[7] Handorean, R., Payton, J., Julien, C., Roman, G.: "Coordination Middleware Supporting Rapid Deployment of Ad Hoc Mobile Systems"; Proc. ICDCS'03, Workshop on Mobile Computing Middleware, IEEE CS Press, Rhode Island USA, 363-368, 2003.

[8] Hauswirth, M., Podnar, I., Decaer, S.: "On P2P Collaboration Infrastructures"; Proc. of WETICE'05, Workshop on Distributed and Mobile Collaboration (DMC), IEEE CS Press, Linkoping University Sweden, 66-71, 2005.

[9] Heinemann, A., Kangasharju, J., Lyardet, F., Mühlhäuser, M.: "iClouds: Peer-to-Peer Information Sharing in Mobile Environments"; Proc. of Euro-Par'03, Lecture Notes in Computer Science 2790, Klagenfurt Austria, 1038-1045, 2003.

[10] Hirsch, F., Kemp, J., Ilkka, J.: "Mobile Web Services: Architecture and Implementation"; Nokia Research Center. John Wiley & Sons Publisher, 2006.

[11] JXTA Project, 2009, https://jxta.dev.java.net/. Last visit: October 2009.

[12] Marques, J., Navarro, L.: "LaCOLLA: A Middleware to Support Self-Sufficient Collaborative Groups"; Computing and Informatics, 25(6), 571-595, 2006.

[13] Mascolo, C., Capra, L., Zachariadis, S., Emmerich, W.: "XMIDDLE: A Data-Sharing Middleware for Mobile Computing"; Journal on Personal and Wireless Communications, 21(1), 77-103, 2002.

[14] Messeguer, R., Ochoa, S.F., Pino, J.A., Medina, E., Navarro, L., Royo, D., Neyem, A.: "Building Real-World Ad-Hoc Networks to Support Mobile Collaborative Applications: Lessons Learned". Lecture Notes in Computer Science, Volume 5784, Springer, 1-16 September 2009.

[15] Monares, A. Ochoa, S.F., Pino, J.A., Herskovic, V., Neyem, A.: "MobileMap: A Collaborative Application to Support Emergency Situations in Urban Areas". Proceedings of the 2009 13th International Conference on Computer Supported Cooperative Work in Design (CSCWD'09), Santiago, Chile, IEEE Press, 565-570, Apr. 2009.

[16] Nemlekar, M.: "Scalable Distributed Tuplespaces"; MSc. Thesis. Department of Electrical and Computer Engineering, North Carolina State University, Chapter 5, 2001.

[17] Neyem, A., Ochoa, S.F., Pino, J.A.: "Integrating Service-Oriented Mobile Units to Support Collaboration in Ad-hoc Scenarios". Journal of Universal Computer Science 14(1), 88-122. 2008.

[18] Neyem, A., Ochoa, S.F., Pino, J.A. "Communication Patterns to Support Mobile Collaboration". Proceedings of the 15th International Workshop on Groupware. Duoro, Portugal, Sept. 13-17, 2009.

[19] Ochoa, S.F., Pino, J.A., Bravo, G., Dujovne, N. Neyem, A.: "Mobile Shared Workspaces to Support Construction Inspection Activities". In P. Zaraté, J.P. Belaud, G. Camilieri, F. Ravat (eds.): Collaborative Decision Making: Perspectives and Challenges, IOS Press, Amsterdam, 270-280, 2008.

[20] Ravichandran, T. and Rothenberger, M. A.: "Software reuse strategies and component markets". Communications ACM 46, 8, 109-114. Aug. 2003.

[21] Rodríguez-Covili, J., Ochoa, S.F., Pino, J.A., Favela, J., Mejía, D., Morán, A.L.: "Designing Mobile Shared Workspaces by Instantiation". Proceedings of the 2009 13th International Conference on Computer Supported Cooperative Work in Design (CSCWD'09), Santiago, Chile, IEEE Press, 402-407. Apr. 2009.

[22] Rodríguez-Covili, J., Ochoa, S.F., Pino, J.A.: "HLMP: High Level MANET Protocol". Technical Report TR/DCC-2009-11, Department of Computer Science, Universidad de Chile. Nov. 2009.

[23] Sadaoui, S. and Yin, P.: "Generalization for component reuse". In Proceedings of the 42nd Annual Southeast Regional Conference. ACM-SE 42. ACM Press, 134-139, April 2004.